Projet De Stijl 2.0 : Plateforme pour robots mobiles

Programmation et conception de systèmes temps réel – 4éme année AE/IR

Institut National des Sciences Appliquées de Toulouse

Cahier des charges fonctionnel

Version 3.0.1 (26 mars 2024)

Référent pédagogique : T. Kloda (d'après P.-E. Hladik) (tomasz.kloda@insa-toulouse.fr)

Référents plateforme : S. Di Mercurio (dimercur@insa-toulouse.fr)

Travail demandé

1. Le travail sera réalisé dans le cadre d'un groupe de deux étudiants sur cinq séances de TP.
2. Le code réalisé devra être rendu sous forme de dépôt GitHub ou d'archive ne comprenant que les codes sources modifiés.
3. La démonstration aura lieu la semaine qui suit la dernière séance de TP.

Ressources indispensables

1. Page Moodle : Programmation et conception sur exécutif temps réel
2. Dépôt GEI-INSA sur GitHub : dumber

1 Vue générale de la plate-forme

Le projet *De Stijl* est une plate-forme de contrôle d'un robot mobile développée au département Génie Électronique et Informatique. Les éléments constituant la plate-forme sont fournis et ont été testés de manière unitaire, mais nous ne garantissons pas un fonctionnement parfait. Toutes suggestions, corrections et modifications seront appréciées pour faire évoluer ce TP. Vous pouvez soumettre vos corrections (sujet, code ou autre) via le dépôt GitHub (branche *evoxx-dumber-v3*).

1.1 Éléments de la plate-forme

La plate-forme est constituée de quatre éléments (voir figure 1) :

1. **L'arène :** Boîte rectangulaire de couleur grise, elle est le terrain dans lequel le robot évolue.
2. **Le robot mobile :** Robot deux roues embarquant un microcontrôleur et un ensemble de composants matériels nécessaires à son déplacement. L'intelligence embarquée est volontairement limitée au contrôle de son déplacement (contrôle des moteurs) et à des fonctionnalités pour connaître son état.
3. **Le superviseur :** Entité principale de la plate-forme, elle est dédiée au contrôle et à la supervision du robot. Elle est couplée à une Webcam fixée au-dessus du terrain afin d'en faire une acquisition visuelle et de localiser le robot.
4. **Le moniteur :** Entité entièrement logicielle et distante, son rôle est d'offrir une interface de contrôle pour l'utilisateur.

La communication entre les équipements est hétérogène et assurée par différents supports :

1. La communication entre le robot et le superviseur est réalisée par une liaison sans fils point à point à l'aide de modules XBee. Cette communication est vue comme une liaison série.
2. La communication entre le superviseur et le moniteur est réalisée par un socket. La liaison physique est assurée par WiFi.

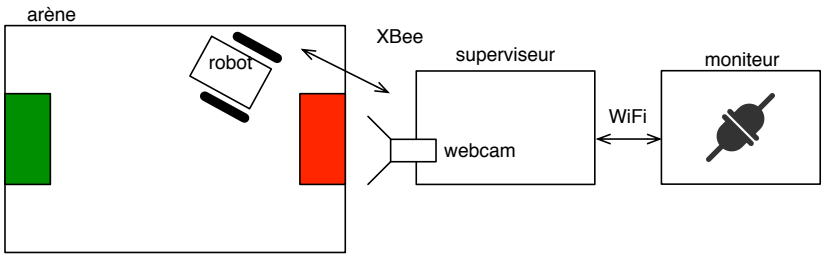

Fig. 1: Vue générale de la plate-forme

1.1.1 Arène

L'arène (voir figure 2) a été réalisée dans les ateliers du département par J. Perez. C'est une boîte de 60×80 cm en Medium et recouverte d'une peinture uniforme grise. Les bords sont peints en blanc. Deux zones distinctes sont peintes sur les bords opposés. L'une étant de couleur verte (ou orange) et l'autre de couleur rouge. Actuellement ces zones ne servent pas.

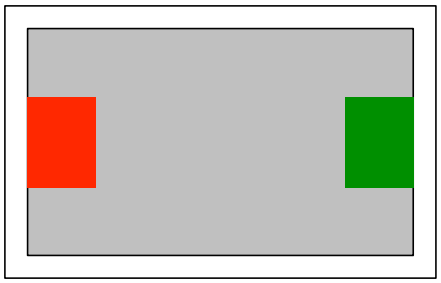

FIG. 2: Vue schématique de l'arène

1.1.2 Robot

La version 2.1 du robot a été conçue et réalisée par S. Di Mercurio. Schématiquement (figure 3), le robot est constitué de deux moteurs, d'un microcontrôleur STM32F103RB et d'une puce Xbee assurant une liaison série point à point avec le superviseur. Chaque robot a un symbol différent sur le dos afin de l'identifier et de le localiser dans l'arène.

Le robot est mobile sur deux roues avec un patin. Le contrôle de la trajectoire est assuré par le contrôle en vitesse des moteurs.

Le code embarqué dans les robots a été produit par S. Di Mercurio et L. Senaneuch. Il offre tous les services nécessaires pour contrôler le robot.

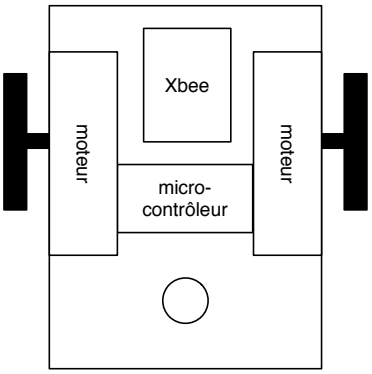

FIG. 3: Vue schématique du robot

Remarque : L'application que vous allez réaliser sera celle du superviseur. Vous ne toucherez pas au code embarqué dans le robot.

1.1.3 Superviseur

Le superviseur orchestre le fonctionnement de la plate-forme en assurant le respect des contraintes temporelles du système. Il est déployé sur une Raspberry Pi 3B sur laquelle est installé un Ubuntu patché PREEMPT-RT avec l'extension mercury de Xenomai 3.0. Une Raspberry Pi est un ordinateur à processeur ARM de taille réduite. La version 3B possède un processeur Broadcom BCM2837 64 bit à quatre cœurs ARM Cortex-A53 à 1,2 GHz, de puces WiFi 802.11n et Bluetooth 4.1. Un module Xbee a été ajouté pour communiquer avec le robot.

La webcam est intégrée au bloc du superviseur et est une caméra Raspberry.

1.1.4 Moniteur

Le moniteur permet à l'utilisateur de saisir les ordres que le robot doit réaliser et aussi de connaître l'état global du système. Il a été développé par S. Di Mercurio en *python*. Le moniteur communique avec le superviseur via un socket. Le serveur est mis en place sur la superviseur et le moniteur en est le client.

1.2 Bibliothèques logicielles

> Votre travail consistera à concevoir uniquement l'architecture logicielle du superviseur. Toutes les fonctions de traitement (communication, vidéo, etc.) **ont déjà été implémentées** par L. Senaneuch et S. Di Mercurio. Vous n'aurez pas à modifier ce code, simplement à faire appel aux fonctions. Il n'en reste pas moins un gros travail d'architecte logiciel à faire.

Le code est disponible sur la branche stage du dépôt GEI-INSA/dumber sur GitHub (branche *evoxx-dumber-v3*). L'annexe B présente sous forme de diagramme de classes les bibliothèques disponibles.

Les fonctions de traitement sont réparties en cinq bibliothèques :
- commonitor : services de communication entre le superviseur et le moniteur,
- comrobot : services de communication entre le superviseur et le robot,
- message : services définissant le format des messages entre le moniteur, le superviseur et le robot,
- camera : services de gestion de la caméra,
- img : services réalisant tous les traitements vidéos.

> Toutes les bibliothèques ont été développées en C++, mais vous n'avez pas besoin de connaissances évoluées pour les utiliser (aucun support ne sera fourni par les encadrants pour le C++). Des explications sont fournies dans un document annexe pour manipuler les bibliothèques, mais d'une manière générale, vous n'avez pas besoin de connaissances évoluées en programmation objet pour les utiliser.

2 Expression fonctionnelle du besoin

2.1 Présentation générale

L'objectif est de concevoir et de développer l'architecture logicielle du superviseur afin d'assurer le fonctionnement de la plate-forme. Les fonctions attendues sont :

1. Assurer la communication entre le moniteur et le superviseur.
2. Assurer la communication entre le superviseur et le robot.
3. Superviser l'état du robot.
4. Contrôler le déplacement du robot.
5. Contrôler et diffuser les images de la webcam.

> **Remarque :** Les plateformes matérielle et logicielle ont de nouveau évoluées cette année. Les bibliothèques fournies ont été réécrites pour faciliter leur prise en main. Bien qu'elles aient été testées, elles comportent certainement des erreurs. Soyez indulgents, nous faisons de notre mieux pour fournir une plate-forme opérationnelle !
> **Si vous remarquez des erreurs, des problèmes ou des optimisations à faire, n'hésitez pas à le communiquer** (via GitHub pour en garder la trace).

2.2 Diagramme de contexte

La figure 4 présente le superviseur dans son contexte et ses interactions avec les autres composants de la plate-forme.

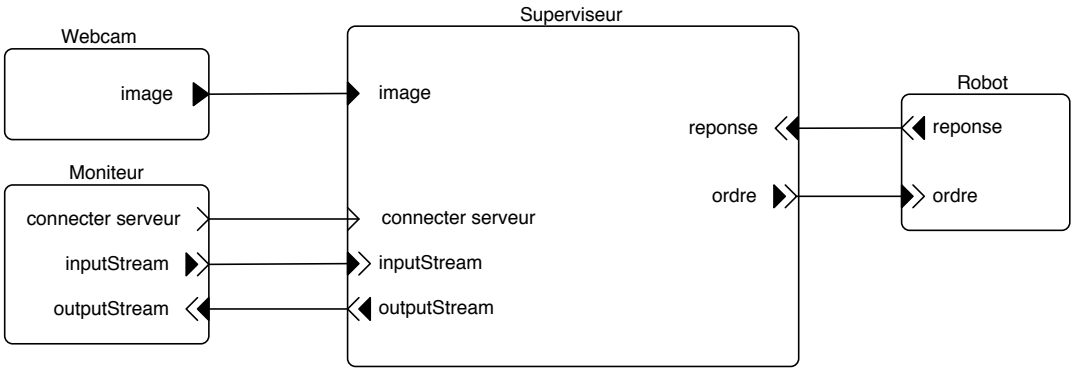

Fig. 4: Diagramme de contexte

La webcam produit des données (`image`) sous la forme d'un tableau d'octets. Le robot reçoit des ordres (`ordre`) sous la forme d'une chaîne de caractères et retourne une réponse aussi sous la forme d'une chaîne (`reponse`). Le moniteur envoie un événement (`connecter serveur`) pour demander la connexion avec le serveur puis établit une communication bi-directionnelle sous la forme d'un flux d'octets (`inputStream` et `outputStream`).

2.3 Description des fonctionnalités

Vous trouverez dans cette partie la description des différentes fonctionnalités attendues. Les annexes apportent des compléments techniques.

L'expression des fonctionnalités est repérée par des encadrés avec un fond gris. C'est ce que vous devez réaliser.

2.3.1 Fonctionnement du moniteur

Le moniteur sert pour le contrôle du robot et la visualisation d'une scène. La communication entre le moniteur et le superviseur est réalisée par un socket avec un serveur du côté du superviseur. La communication est bi-directionnelle : le moniteur peut envoyer des requêtes et le superviseur peut envoyer des données pour mettre à jour certains éléments de l'interface. Les messages sont prédéfinis et sont présentés dans l'annexe A.2.

2.3.2 Communication entre le superviseur et le moniteur

La communication entre le serveur et le superviseur est réalisée à l'aide d'un socket. Le superviseur joue le rôle de serveur. Toutes les fonctions pour gérer la communication entre le moniteur et le superviseur sont fournies dans `monitor`.

Les figures 5 et 6 illustrent le mode de fonctionnement nominal attendu pour la communication entre le superviseur et le moniteur.

Lancement du serveur. Le lancement du serveur est réalisé à l'aide de la méthode `Open` de la classe `ComMonitor`.

Fonctionnalité 1 : Le lancement du serveur doit être réalisé au démarrage du superviseur. En cas d'échec du démarrage du serveur, un message textuel doit être affiché sur la console de lancement de l'application. Ce message doit signaler le problème et le superviseur doit s'arrêter.

Etablissement du socket. La connexion entre le moniteur et le superviseur est réalisée suite à la demande de l'utilisateur via l'interface graphique. Lorsque la demande est faite, un socket est créé, il faut donc que le serveur soit en attente d'une demande de connexion, c'est-à-dire que la méthode `AcceptClient` de la classe `ComMonitor` soit en cours d'exécution. Cette méthode est bloquante.

Fonctionnalité 2 : La connexion entre le moniteur et le superviseur (via le socket) doit être établie suite à la demande de connexion de l'utilisateur.

Réception des messages. La réception des messages du serveur par le superviseur est réalisée par l'appel de la méthode `Read` de la classe `ComMonitor`. Cette méthode est bloquante. Les messages du moniteur vers le superviseur sont pré-formatés dans la classe `Message` et définis dans l'annexe A.2.1.

Fonctionnalité 3 : Tous les messages envoyés depuis le moniteur doivent être réceptionnés par le superviseur.

Envoi des messages. L'envoi des messages du superviseur vers le moniteur est réalisé à l'aide de la fonction `Write` de la classe `ComMonitor`. Les messages du superviseur vers le moniteur sont pré-formatés et définis dans l'annexe A.2.2.

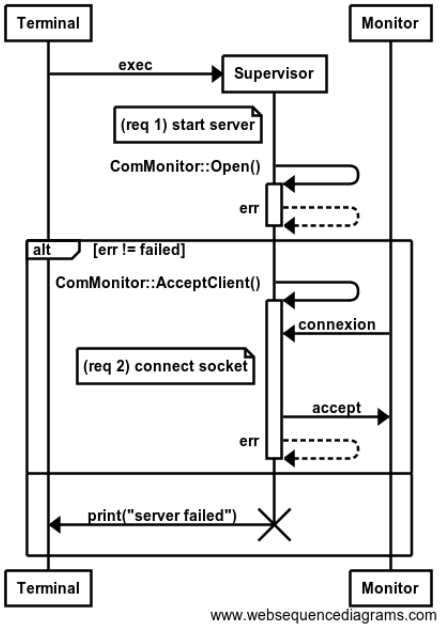

FIG. 5: Diagramme de séquence des fonctionnalités 1 à 2

Fonctionnalité 4 : L'application superviseur doit être capable d'envoyer les messages au moniteur (via le serveur) avec un délai d'au plus 10 ms.

Détection de la perte de communication. La perte de communication entre le moniteur et le superviseur est détectée lors de la lecture sur le socket. La méthode `Read` retourne un message de type `MESSAGE_MONITOR_LOST` si la communication est perdue.

Fonctionnalité 5 : Le superviseur doit détecter la perte de communication avec le moniteur. En cas de perte de la communication un message doit être affiché sur la console de lancement du superviseur.

Reprise de la communication. La communication entre le moniteur et le superviseur est fermé à l'aide de la méthode `Close` de la classe `ComMonitor`. **Attention** cette fonctionnalité ne peut être mise en place que lorsque l'ensemble de l'application est déjà réalisée, il ne sert à rien d'y travailler tant que le reste n'est pas fait.

Fonctionnalité 6 : En cas de perte de communication entre le superviseur et moniteur, il faut stopper le robot, la communication avec le robot, fermer le serveur et déconnecter la caméra afin de revenir dans le même état qu'au démarrage du superviseur.

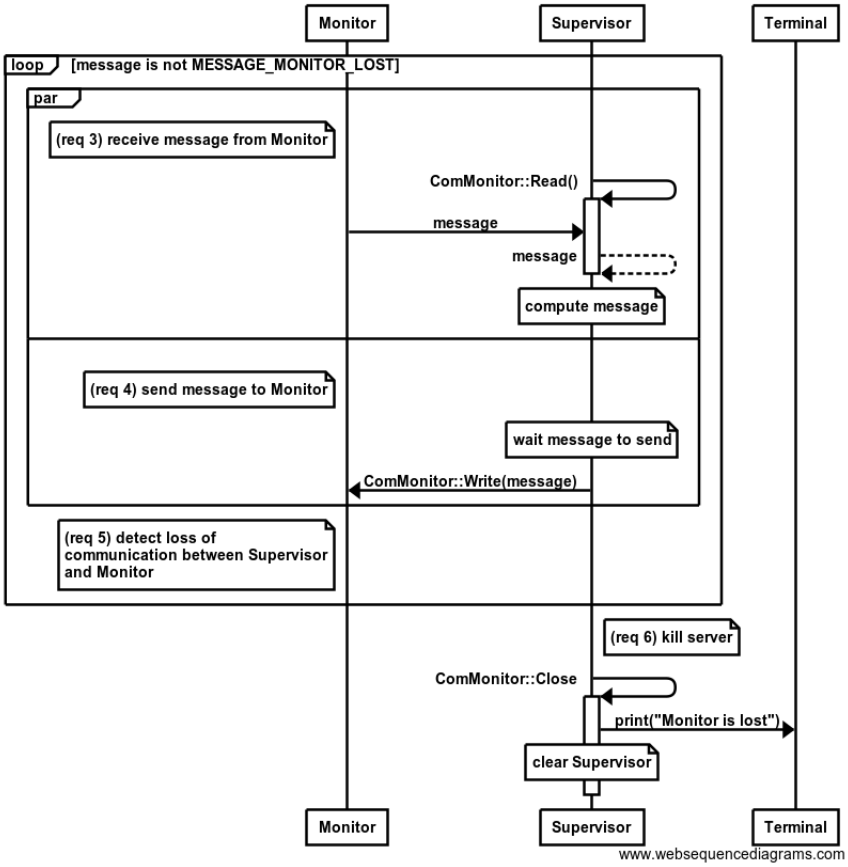

FIG. 6: Diagramme de séquence des fonctionnalités 3 à 6

2.3.3 Communication entre le superviseur et le robot

La communication avec le robot s'effectue par le biais d'émetteur-récepteur Xbee. Pour communiquer, il est nécessaire de commencer par ouvrir la communication entre le superviseur et le boîtier Xbee. Toutes les fonctions de gestion de la communication entre le robot et le superviseur sont fournies dans la bibliothèque `robot`. La figure 7 illustre les fonctionnalités 7 à 9.

Mettre en place la communication avec le robot. L'ouverture de la communication (c'est-à-dire la réservation d'un port série) avec le robot se fait à l'aide de la fonction `Open` de la classe `ComRobot`. Cette demande est automatiquement demandée par le moniteur dès que le socket est en place.

Fonctionnalité 7 : Dès que la communication avec le moniteur est en place, la communication entre le superviseur et le robot doit être ouverte. Si la communication est active, il faut envoyer un message d'acquittement au moniteur. En cas d'échec, un message le signalant est renvoyé au moniteur.

Surveillance de la communication avec le robot. La communication entre le robot et le superviseur peut être perdue. Afin de surveiller cela, il faut mettre en place un mécanisme permettant d'inférer cette perte. L'envoi des messages au robot se fait à l'aide de la fonction `Write`

de la classe `ComRobot`. En cas d'échec de communication (`MESSAGE_ANSWER_ROBOT_TIMEOUT`), la méthode retourne un message contenant un code d'erreur. Cependant, l'envoi d'un message par Xbee peut retourner un échec même si le médium de communication est encore opérationnel.

De ce fait, le simple retour d'un échec ne suffit pas à déterminer si la communication est définitivement perdue ou bien si c'est une erreur fugace. Afin de conclure que la communication est perdue, il faut donc mettre en place un mécanisme de compteur. Pour cela, il faut incrémenter un compteur à chaque échec sur l'envoi d'un message et le remettre à zéro à chaque succès. Si le compteur dépasse 3, la communication est alors déclarée perdue.

Fonctionnalité 8 : La communication entre le robot et le superviseur doit être surveillée par un mécanisme de compteur afin de détecter une perte du médium de communication.

Perte de la communication avec le robot. Le medium de communication entre le robot et le superviseur est stoppé par l'appel à la fonction `close_communication_robot`.

Fonctionnalité 9 : Lorsque la communication entre le robot et le superviseur est perdue, un message spécifique doit être envoyé au moniteur. Le système doit fermer la communication entre le robot et le superviseur et se remettre dans un état initial permettant de relancer la communication.

2.3.4 Démarrage du robot

Tous les messages entre le superviseur et le robot sont décrits dans l'annexe A.1. L'envoi d'un message se fait par l'utilisation de la méthode `Write` de la classe `ComRobot`. La figure 8 représente la séquence pour les fonctionnalités 10 et 11.

Remarque : Aucun mécanisme n'est mis en œuvre dans l'implémentation de `Write` de `ComRobot` pour se prémunir des appels concurrents.

Démarrage sans watchdog du robot. Le robot a deux modes de démarrage. Un mode simple, dit sans watchdog et un mode évolué exposé ci-après.

Fonctionnalité 10 : Lorsque l'utilisateur demande, via le moniteur, le démarrage sans watchdog, le robot doit démarrer dans ce mode. En cas de succès, un message d'acquittement est retourné au moniteur. En cas d'échec, un message indiquant l'échec est transmis au moniteur.

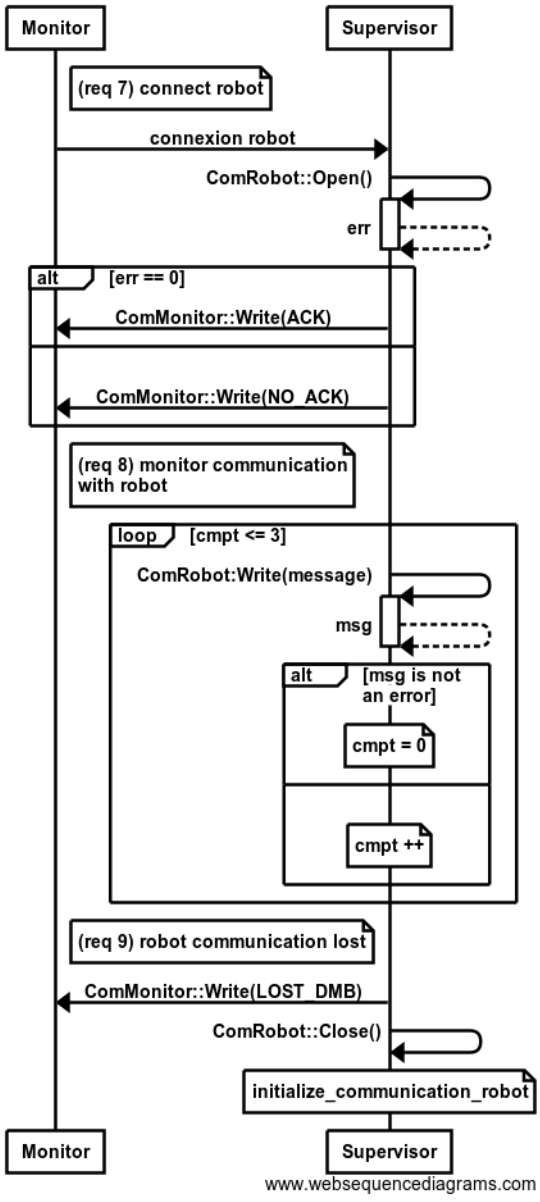

Fig. 7: Diagramme de séquence des fonctionnalités 7 à 9

Démarrage avec watchdog du robot. En cas de perte de la communication entre le superviseur et le robot, ce dernier n'est plus contrôlable et continue à exécuter des ordres. Pour éviter cela, un mécanisme de surveillance à base de watchdog a été mis en place sur le robot.

Le principe est simple : au démarrage du robot (c.-à-d. quand le robot traite l'ordre de démarrage avec watchdog), un watchdog périodique d'une seconde est lancé. À chaque expiration du watchdog un compteur dans le robot est incrémenté. Si le compteur atteint 3, le robot s'arrête et doit être redémarré manuellement. Pour éviter cela, le robot doit recevoir du superviseur un ordre spécifique de rechargement du watchdog (la fonction ReloadWD de ComRobot). Si l'ordre est valide, le compteur est décrémenté de 1 (minimum 0). Pour que cet ordre soit valide il faut qu'il arrive au moment où le watchdog expire avec un tolérance de 50 ms.

Fonctionnalité 11 : Lorsque l'utilisateur demande, via le moniteur, le démarrage avec watchdog, le robot doit démarrer dans ce mode. Un message d'acquittement est retourné au moniteur. En cas d'échec, un message indiquant l'échec est transmis au moniteur.
Une fois le démarrage effectué, le robot doit rester vivant. Pour cela, il faut que le moniteur envoie régulièrement le message de rechargement du watchdog.

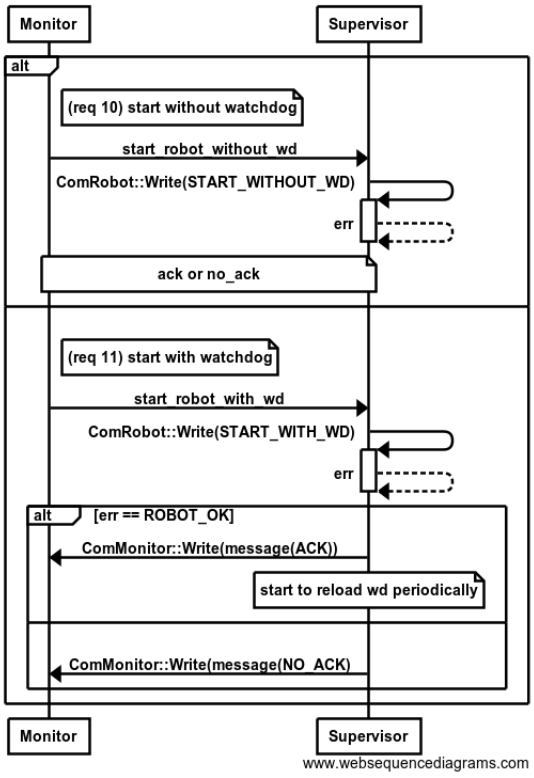

FIG. 8: Diagramme de séquence des fonctionnalités 10 et 11

2.3.5 Déplacement et état du robot

Le robot n'a aucune intelligence et ne réagit qu'aux ordres qu'il reçoit. La figure 9 représente la séquence pour les fonctionnalités 12 et 13.

Déplacement manuel du robot. Le robot peut recevoir cinq ordres de mouvement (avancer, reculer, tourner à droite, tourner à gauche et stopper). Lorsque l'utilisateur presse les flèches de l'interface graphique un message est envoyé au superviseur avec le mouvement à réaliser.

Fonctionnalité 12 : Lorsque qu'un ordre de mouvement est reçu par le superviseur, le robot doit réaliser ce déplacement en moins de 100 ms.

Niveau de batterie du robot. Il est possible de récupérer le niveau de la batterie du robot à l'aide de la fonction `Write` de `ComRobot` avec le message `getBattery` de `ComRobot` aussi. La valeur retournée est ensuite à transmettre au moniteur.

Fonctionnalité 13 : Le niveau de la batterie du robot doit être mis à jour toutes les 500 ms sur le moniteur.

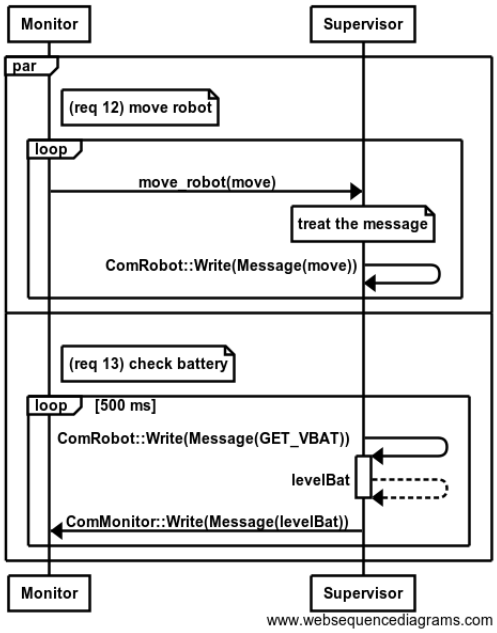

FIG. 9: Diagramme de séquence des fonctionnalités 12 et 13

2.3.6 Gestion de la caméra

Les fonctions permettant la manipulation des images sont fournies dans les librairies `img` et `camera`. L'implémentation des méthodes utilise OPENCV, une libraire libre en C/C++ de traitement d'images. La caméra peut être instantanée avec `camera = new Camera(sm,5)` où `Camera * camera`.

Ouverture de la caméra. La méthode `Open` de la classe `Camera` donne accès à la caméra.

Fonctionnalité 14 : La caméra doit être démarrée suite à une demande provenant du moniteur. Si l'ouverture de la caméra a échoué, il faut envoyer un message au moniteur.

Capture d'une image (mode nominal). La méthode `Grab` de `Camera` permet de capturer une image. L'envoi de l'image au moniteur s'effectue normalement par l'envoi d'un message [1]. La fréquence de capture d'une image est fixée par un paramètre lors de l'instanciation d'un objet `Camera`.

Fonctionnalité 15 : Dès que la caméra est ouverte, une image doit être envoyée au moniteur toutes les 100 ms.

Fermeture de la caméra. La méthode `Close` de la classe `Camera` permet de fermer proprement la caméra.

Fonctionnalité 16 : La caméra doit être fermée suite à une demande provenant du moniteur. Un message doit être envoyé au moniteur pour signifier l'acquittement de la demande. L'envoi périodique des images doit alors être stoppé.

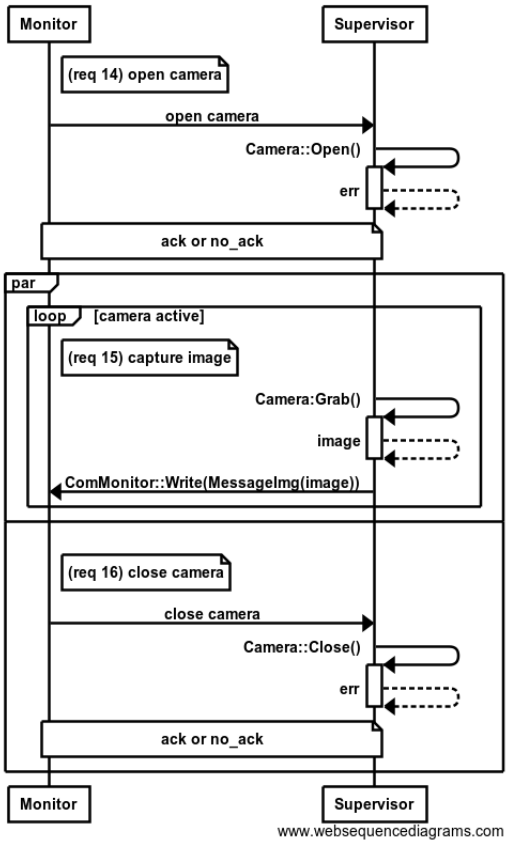

FIG. 10: Diagramme de séquence des fonctionnalités 14 et 16

Calibration de l'arène Pour accélérer le traitement de l'image et le rendre plus stable [2], il est préférable de limiter la zone d'étude à l'arène. Pour cela il est nécessaire de faire un pré-traitement

1. L'image est compressée lors de l'envoi du message.
2. Si vous placez un robot en dehors de l'arène, il y a de fortes chances pour que cela perturbe le calcul de la position du robot.

qui recherche l'arène.

Le calcul se fait par l'appel à la méthode `SearchArena` de `Img`. Il est possible de tracer sur l'image l'arène en faisant appel à la méthode `DrawArena`.

Fonctionnalité 17 : Suite à une demande de recherche de l'arène, le superviseur doit stopper l'envoi périodique des images, faire la recherche de l'arène et renvoyer une image sur laquelle est dessinée cette arène. Si aucune arène n'est trouvée un message d'échec est envoyé.

L'utilisateur doit ensuite valider visuellement via le moniteur si l'arène a bien été trouvée. L'utilisateur peut :

- valider l'arène : dans ce cas, le superviseur doit sauvegarder l'arène trouvée (pour l'utiliser ultérieurement) puis retourner dans son mode d'envoi périodique des images en ajoutant à l'image l'arène dessinée.
- annuler la recherche : dans ce cas, le superviseur doit simplement retourner dans son mode d'envoi périodique des images et invalider la recherche.

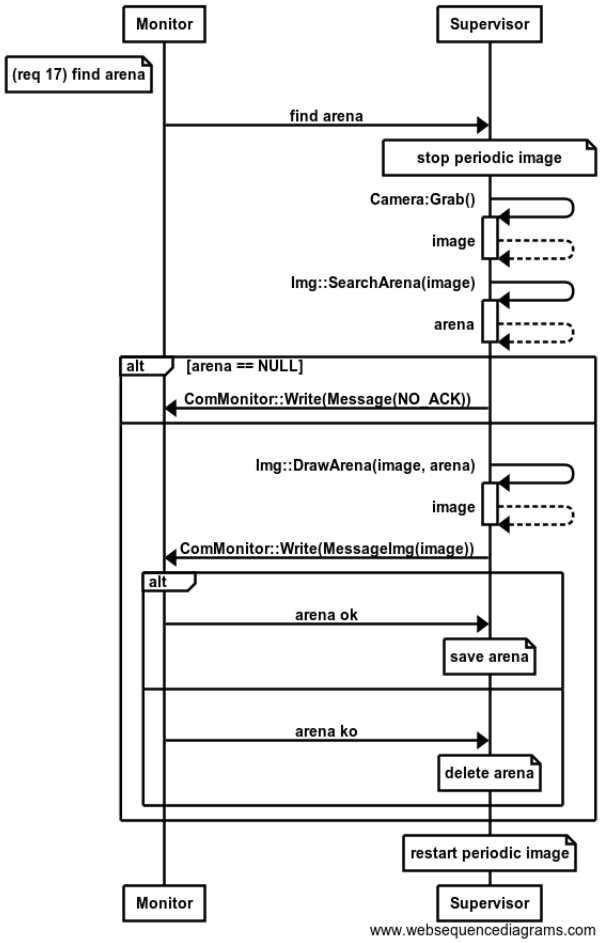

Fig. 11: Diagramme de séquence de la fonctionnalité 17

Calcul de la position du robot. Le traitement d'une image pour trouver la position du robot se fait à l'aide de la méthode `SearchRobot` de `Img`. Il est possible de dessiner sur l'image la position trouvée en faisant appel à la méthode `DrawRobot`. La position est envoyée du superviseur vers le moniteur en utilisant un message avec une entête dédié.

Fonctionnalité 18 : Suite à une demande de l'utilisateur de calculer la position du robot, le superviseur doit calculer cette position, dessiner sur l'image le résultat et envoyer un message au moniteur avec la position toutes les 100 ms. Si le robot n'a pas été trouvé, un message de position est envoyé avec une position (-1,-1).

Stopper le calcul de la position du robot. Il est possible pour l'utilisateur de demander l'arrêt du calcul de la position.

Fonctionnalité 19 : Suite à une demande de l'utilisateur de stopper le calcul de la position du robot, le superviseur doit rebasculer dans un mode d'envoi de l'image sans le calcul de la position.

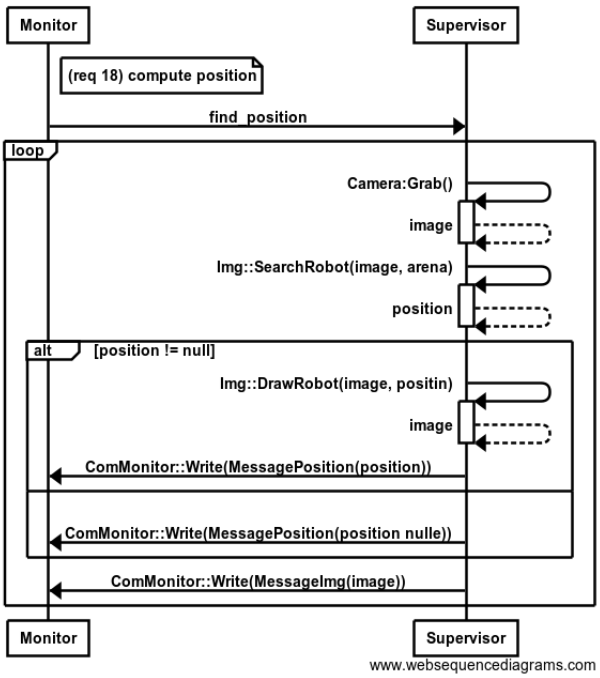

Fig. 12: Diagramme de séquence de la fonctionnalité 18

Annexes

A Annexes

A.1 Messages robot-superviseur

La communication fonctionne sur le principe d'une communication synchrone de type requête. La communication ne peut être initiée que par le superviseur.

Toutes les communications ont par défaut un retour en cas d'erreur. Ce retour est porté par la valeur de `messageID` d'un objet `Message`. Si la communication s'est bien déroulée la valeur MESSAGE_ANSWER_ACK est retournée, sinon une valeur de retour correspondante à l'un des cas d'erreur suivant est produite :

- MESSAGE_ANSWER_ROBOT_TIMEOUT : la réponse n'est pas arrivée avant 80 ms,
- MESSAGE_ANSWER_ROBOT_UNKNOWN_COMMAND : la commande n'a pas été comprise par le robot,
- MESSAGE_ANSWER_ROBOT_ERROR : la commande n'est pas conforme à sa définition.

Les messages envoyés au robot sont composés d'une entête codée sur un octect (`char`) correspondant à l'ordre à réaliser. Certains ordres sont aussi accompagnés d'une donnée codée sur un entier (`int`). Le tableau 1 décrit les entêtes.

Entête (MESSAGE_ROBOT_)	Données	Description	Retour
PING	–	Teste la disponibilité de la communication	–
RESET	–	Demande un redémarrage du robot	–
START_WITHOUT_WD	–	Démarre le robot sans le *watchdog*	–
START_WITH_WD	–	Démarre le robot avec le *watchdog*	–
RELOAD_WD	–	Recharge le *watchdog*	–
BATTERY_GET	–	Retourne le niveau de la batterie	niv. batterie
STATE_GET	–	Retourne l'état du robot	état
GO_FORWARD	–	Déplace le robot en avant	–
GO_BACK	–	Déplace le robot en arrière	–
GO_LEFT	–	Fait tourner dans le sens anti-horaire	–
GO_RIGHT	–	Fait tourner dans le sens horaire	–
STOP_MOVE	–	Stoppe le mouvement du robot	–
MOVE	distance	Déplace en ligne droite	–
TURN	angle	Tourne d'un angle donné	–

TAB. 1: Liste des messages du superviseur vers le robot

Le niveau de la batterie peut prendre comme valeur :

- BATTERY_UNKNOWN : la mesure n'a pas abouti,
- BATTERY_EMPTY : niveau vide,
- BATTERY_LOW : niveau faible,
- BATTERY_FULL : niveau haut.

Les états possibles du robot sont :

- MESSAGE_ROBOT_STATE_BUSY : le robot est en train de réaliser un mouvement,
- MESSAGE_ROBOT_STATE_NOT_BUSY : le robot ne réalise pas de mouvement.

A.2 Messages moniteur-superviseur

A.2.1 Moniteur vers Superviseur

Les message envoyés du moniteur vers le superviseur sont composés d'un entête de trois octets et d'une donnée d'un octet. Le tableau 2 décrit ces entêtes et les données. Certains messages nécessitent une réponse par un acquittement (cf. messages section A.2.2). Les messages spécifiques pour le robot ont les même entêtes que la communication entre le robot et le superviseur.

Id	**Description**
MESSAGE_ROBOT_COM_OPEN	Demande d'ouverture de la comm. avec le robot
MESSAGE_ROBOT_COM_CLOSE	Demande de fermeture de la comm. avec le robot
Acquittement : oui	
MESSAGE_ROBOT_*	Message portant un ordre pour le robot (voir tableau précédent)
Acquittement : les messages MESSAGE_ROBOT_START_WITH_WD et MESSAGE_ROBOT_START_WITHOUT_WD nécessitent un acquitement	
MESSAGE_CAM_OPEN	Demande d'ouverture de la caméra
MESSAGE_CAM_CLOSE	Demande de fermeture de la caméra
MESSAGE_CAM_ASK_ARENA	Demande de détection de l'arène
MESSAGE_CAM_ARENA_CONFIRM	L'arène est la bonne
MESSAGE_CAM_ARENA_INFIRM	L'arène n'est pas la bonne
MESSAGE_CAM_POSITION_COMPUTE_START	Calcul de la position du robot
MESSAGE_CAM_POSITION_COMPUTE_STOP	Arrêt du calcul de la position du robot
Acquittement : les messages MESSAGE_CAM_OPEN et MESSAGE_CAM_CLOSE attendent un acquittement.	

Tab. 2: Description des messages du moniteur vers le superviseur

A.2.2 Superviseur vers Moniteur

Les messages du superviseur vers le moniteur sont composés d'une entête de 3 octets et de données de taille variables. Le tableau 3 décrit ces messages.

Entête	Données	Description
MESSAGE_ANSWER_ACK	–	Message d'acquittement en cas de succès
MESSAGE_ANSWER_NACK	–	Message d'acquittement en cas d'échec
MESSAGE_MONITOR_LOST	–	Signale la perte de la comm. avec le robot
MESSAGE_CAM_IMAGE	Image	Envoi d'une image
MESSAGE_CAM_POSITION	Position	Envoi de la postion du robot
MESSAGE_ROBOT_BATTERY_LEVEL	int	Envoi du niveau de la batterie

TAB. 3: Liste des messages du moniteur vers le superviseur

B Diagramme de classe des bibliothèques

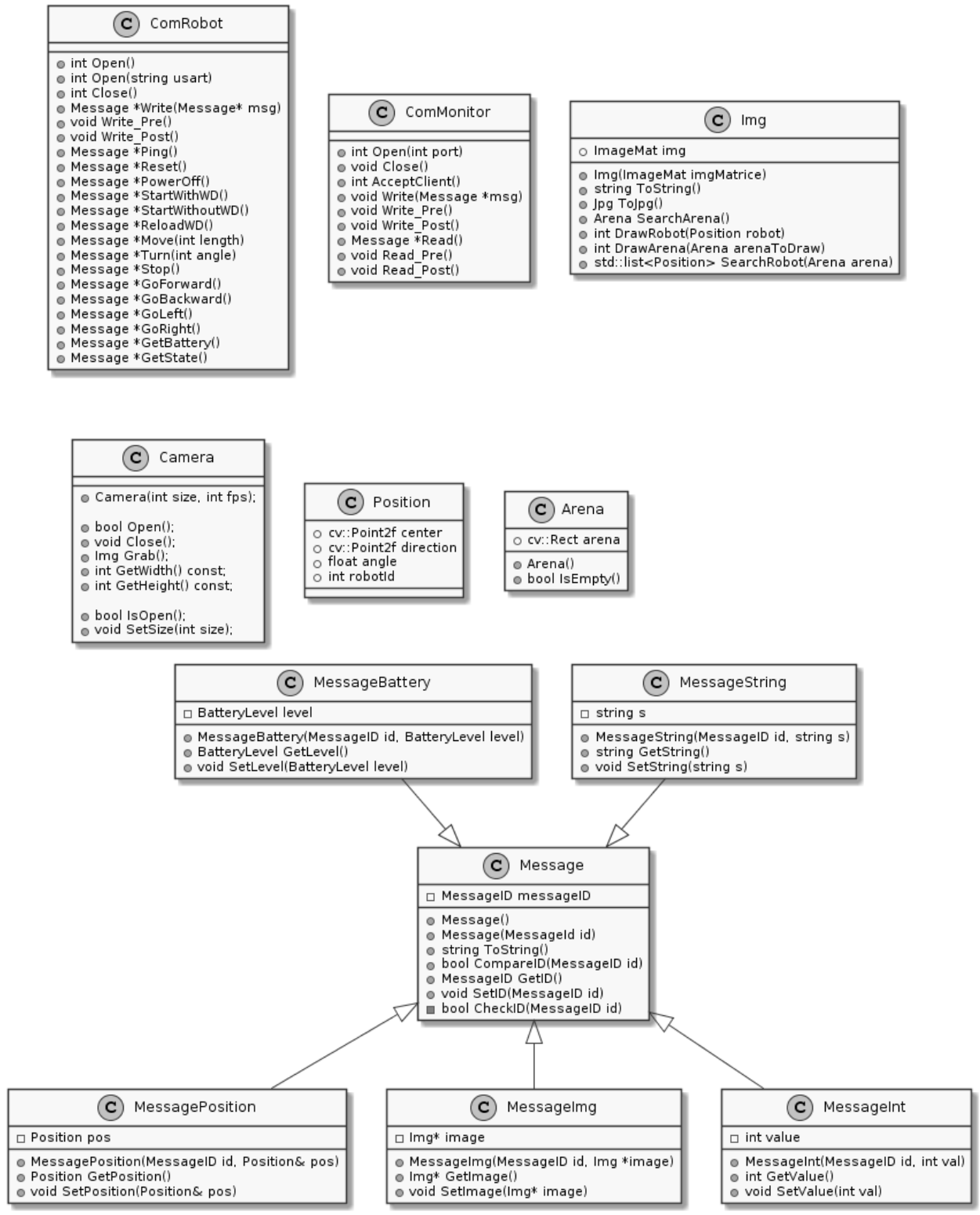

FIG. 13: Diagramme de classes des bibliothèques De Stijl